UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Herramienta para la visualización y análisis de datos heterogéneos sobre la ciudad de Madrid

Adrián Montero Masedo
Tutor: Iván Cantador Gutiérrez

JULIO 2020

Herramienta para la visualización y análisis de datos heterogéneos sobre la ciudad de Madrid

AUTOR: Adrián Montero Masedo
TUTOR: Iván Cantador Gutiérrez

Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio de 2020

Resumen

En los últimos años las tecnologías de la información han sufrido una gran evolución, gracias a los avances en hardware, software y telecomunicaciones, que han permitido mejorar la gestión de los datos, aumentar el volumen de los mismos, y facilitar su acceso. Esta evolución junto con la demanda ciudadana de transparencia por parte de las instituciones públicas ha propiciado que las administraciones gubernamentales implementen iniciativas de publicación de datos abiertos.

Los datos abiertos son datos de acceso libre y reutilizables, en general bien estructurados y en formatos estandarizados, que carecen de derechos de autor o restricciones de patentes. Un ejemplo puede ser el portal de datos abiertos del Ayuntamiento de Madrid, en el que se publican, entre otros, datos sobre las actuaciones del SAMUR, calidad del aire o la recaudación vinculada a los parquímetros en la ciudad.

Estos datos suelen estar recogidos en bases de datos o ficheros de representación de datos en formato de tablas, como pueden ser CSV o XLSX, que requieren de un gran esfuerzo para ser interpretados por una persona.

En este Trabajo de Fin de Grado se propone el desarrollo de una aplicación Web que permita la visualización de datos y estadísticas recopiladas por las plataformas de datos abiertos de Madrid, facilitando el análisis de los mismos mediante el uso de gráficas, nubes de palabras o cualquier otro método que simplifique la comprensión y análisis, haciendo que su interpretación sea sencilla y rápida.

En particular, para la realización de este proyecto se han empleado los datos abiertos de la plataforma electrónica de participación ciudadana, Decide Madrid, en la que se proponen, debaten y votan propuestas presupuestarias para el Ayuntamiento de Madrid.

Palabras clave (castellano)

Datos abiertos, Decide Madrid, transparencia, visualización de datos, HTML, Angular, TypeScript, JAVA, Spring, MySQL.

Abstract

In recent years, information technologies have entailed a great evolution in terms of advances in hardware, software and telecommunications, making it possible to improve data management, increase the volume of processed data, and facilitate access to information. This evolution together with the citizens' demand for transparency from public institutions has led to the implementation of open data publication initiatives by government administrations.

Open data is freely accessible and reusable data, generally well-structured and in standardized formats, which is free of copyright or patent restrictions. One example of such initiatives is the Madrid City Council's Open Data portal, which publishes, among other things, collection on data about SAMUR's actions, air quality, and parking areas in the city.

These data are usually collected in databases or data representation files following a table format, such as CSV or XLSX, requiring a high effort to be interpreted by a person.

This Degree Final Project proposes the development of a web application that allows for the visualization of data and statistics reported in the open data collections of Madrid, facilitating their analysis through the use of graphs, word clouds, and any other method that simplifies the information understanding and analysis, making its interpretation simple and fast.

In particular, we have used open data from Decide Madrid, the electronic platform for citizen participation, where citizens' budget proposals for the Madrid City Council are given, debated and voted.

Keywords

Open data, Decide Madrid, transparency, data visualization, HTML, Angular, TypeScript, JAVA, Spring, MySQL.

Agradecimientos

Quiero agradecer a Iván Cantador haberme dado la oportunidad de realizar este proyecto y guiarme cuando me ha sido necesario.

También quiero dar las gracias a mis padres y mi hermana, gracias por apoyarme en todo momento.

Por último, dar las gracias a Celia, Álvaro, César, Rober e Ibarra por haberme acompañado en esta aventura.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

1 Introducción

1.1 Motivación

Los datos abiertos son, según Open Knowledge Foundation, "*datos que pueden ser utilizados, reutilizados y redistribuidos libremente por cualquier persona, y que se encuentran sujetos, cuando más, al requerimiento de atribución y de compartirse de la misma manera en que aparecen"* [1]. Existe una serie de restricciones en la publicación de estos datos para que puedan denominarse datos abiertos, que son las siguientes:

- Los datos deben ser de fácil acceso, preferiblemente a través de internet, y encontrarse como una entidad.
- Los datos deben de poder ser reutilizados o modificados y deben de poder ser redistribuidos.
- No se puede restringir el acceso a los datos. No pueden existir restricciones sobre los datos y no debe haber discriminación alguna.

En los últimos años, distintos gobiernos municipales, regionales o estatales han optado por la publicación de este tipo de datos, ya sea por motivos de transparencia institucional, fomento de la innovación tecnológica o cualquier otro motivo.

El Ayuntamiento de Madrid posee un portal web en el que publica colecciones de datos abiertos [2] sobre multitud de asuntos como pueden ser los accidentes de tráfico, el consumo de energía en edificios municipales o la deuda del consistorio. La mayoría de estos datos, de acceso mediante su página web, están almacenados en ficheros, de formato abierto, con el fin de ser usados por programas informáticos, no de ser interpretados directamente por personas.

Este proyecto pretende simplificar la lectura e interpretación de datos heterogéneos sobre la ciudad de Madrid, realizando una aplicación que haga uso de elementos visuales, como por ejemplo gráficas o nubes de palabras, para facilitar el análisis de los datos. También se pretende realizar una herramienta sencilla e intuitiva para que pueda ser usada por cualquier persona, independientemente de sus habilidades informáticas.

1.2 Objetivos

El objetivo principal de este proyecto es la creación de una aplicación web que facilite la lectura y análisis de los datos recogidos en la plataforma de datos abiertos del ayuntamiento de Madrid.

El usuario de esta herramienta debe de ser capaz de usarla de manera sencilla e intuitiva y poder visualizar fácilmente los datos deseados. Para ello se han establecido una serie de requisitos.

Requisitos funcionales (RF)

RF-1	Lectura de datos desde una base de datos.
RF-2	Visualización de los datos en tablas.
RF-3	Visualización de los datos en gráficas.
RF-4	Visualización de los datos en nubes de palabras.

Requisitos no funcionales (RNF)

RNF-1 Facilidad para trabajar con distintos modelos de datos.

RNF-2 Facilidad de uso mediante una interfaz de usuario sencilla e intuitiva.

RNF-3 Eficiencia. Tiempos de respuesta y consumo de recursos (memoria, procesador o red) reducidos.

RNF-4 Uso de software libre.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Capítulo 1 – Introducción**

  Motivación y objetivos del proyecto.

- **Capítulo 2 – Estado del arte**

  Tecnologías empleadas actualmente, ejemplos de su uso.

- **Capítulo 3 – Diseño**

  Descripción del planteamiento del proyecto.

- **Capítulo 4 – Desarrollo**

  Detalle del desarrollo del proyecto.

- **Capítulo 5 – Integración, pruebas y resultados**

  Explicación de las pruebas efectuadas y descripción del proceso de integración realizado.

- **Capítulo 6 – Conclusiones y trabajo futuro**

  Conclusiones finales y planificación de futuros desarrollos.

2 Estado del arte

En este capítulo se describirán brevemente las tecnologías empleadas en la actualidad en el desarrollo de aplicaciones web.

2.1 Aplicaciones web

En la actualidad existe una gran cantidad de empresas y entidades que permiten a los usuarios el empleo de sus herramientas mediante el acceso por un navegador web. Las administraciones públicas, como la Agencia Tributaria; redes sociales, como Twitter; cadenas de supermercados, como Carrefour; o medios de comunicación, como RTVE, hacen uso de esta tecnología.

2.1.1 Tecnologías empleadas en el cliente

En la actualidad, las aplicaciones web han adoptado, casi en su totalidad, el uso de JavaScript para hacer la interfaz de usuario dinámica. Actualmente existen una cantidad significativa de frameworks basados en JavaScript, como pueden ser Angular [4], React [10] o Vue [11]. Su principal cualidad es facilitar la creación de componentes web [12] autocontenidos y reutilizables, cuyo modo de empleo es mediante el uso de etiquetas HTML personalizadas. Estos frameworks han permitido que las interfaces de usuario fuesen más dinámicas haciendo que el usuario pueda interaccionar con los distintos elementos que la forman. Además, permiten liberar de carga de trabajo al servidor.

2.1.2 Tecnologías empleadas en el servidor

Por otro lado, las tecnologías usadas en el servidor son muy diversas. Existen infraestructuras de servidor como LAMP [13] y también existen frameworks como Microsoft .NET [14] y JavaEE [15]. Aunque en la actualidad infraestructuras tipo LAMP son muy utilizadas, tecnologías como Microsoft .NET o JavaEE facilitan la labor del desarrollador, ya que simplifican la configuración del entorno de trabajo y la gestión de los elementos que integran el servidor.

LAMP consiste en el empleo de Linux, como sistema operativo del servidor; Apache Tomcat, como servidor web; MySQL o MariaDB como gestores de bases de datos; y PHP como lenguaje de programación del servidor.

El framework desarrollado por Microsoft, .NET, actualmente puede emplearse en servidores que empleen como sistema operativo Windows, Linux y macOS. Este framework permite desarrollar el código del servidor web en múltiples lenguajes como pueden ser C#, C++, Pearl o Python.

JavaEE, al tratarse de una plataforma Java, puede usarse en cualquier servidor, independientemente de su sistema operativo, mientras este sea capaz de ejecutar la máquina virtual de Java. Este framework proporciona un API y un entorno de ejecución para desarrollar y ejecutar aplicaciones en red de gran escala, multi-capa, escalables, fiables y seguras, pero estas características hacen que su diseño sea complejo. Existen gran cantidad de herramientas, desarrolladas por terceros, que amplían la funcionalidad inicial de JavaEE y simplifican su desarrollo. Entre estas herramientas se encuentra Spring [16]. Spring es un framework de desarrollo de aplicaciones que automatiza parte del código mediante el uso de anotaciones.

3 Diseño

Este proyecto se ha diseñado como una aplicación web que sea accesible desde los navegadores web. Basándome en la arquitectura tipo de las aplicaciones web y en las características de las tecnologías empleadas, la arquitectura de este proyecto puede verse en la figura 2.

Figura 1 – Diagrama de arquitectura

3.1 Capa del navegador

El framework Angular 2+, enfocado en el patrón Modelo-Vista-Controlador, se caracteriza por la modularidad de sus componentes, que permiten a los desarrolladores generar módulos con funcionalidades y vista exclusivas, reutilizables en toda la aplicación, con multitud de posibilidades, como pueden ser la inyección de datos provenientes del componente padre y la emisión de eventos hacía el componente padre para provocar algún tipo de respuesta en él. Angular también favorece la carga del código necesario para mostrar la pantalla deseada, lo que mejora su rendimiento. Por otra parte, su capacidad de enrutamiento de componentes posibilita una navegación tradicional. Todo esto evita que se tenga que repetir código y funcionalidad a lo largo del desarrollo. Por todo esto, he decidido diseñar el proyecto siguiendo el diagrama de clases de la figura 2 separando en diferentes componentes por su funcionalidad o por su implicación en la navegación deseada y agrupando por otro lado los servicios, quedando distribuidos de la siguiente manera:

- Home
- Decide-madrid
- Decide-madrid-data
- Decide-madrid-statistics
- Trends
- Open-data
- Nav-menu
- Custom-chart
- Custom-dialog
- Custom-table
- Custom-tag-cloud
- Custom-tweets
- Services

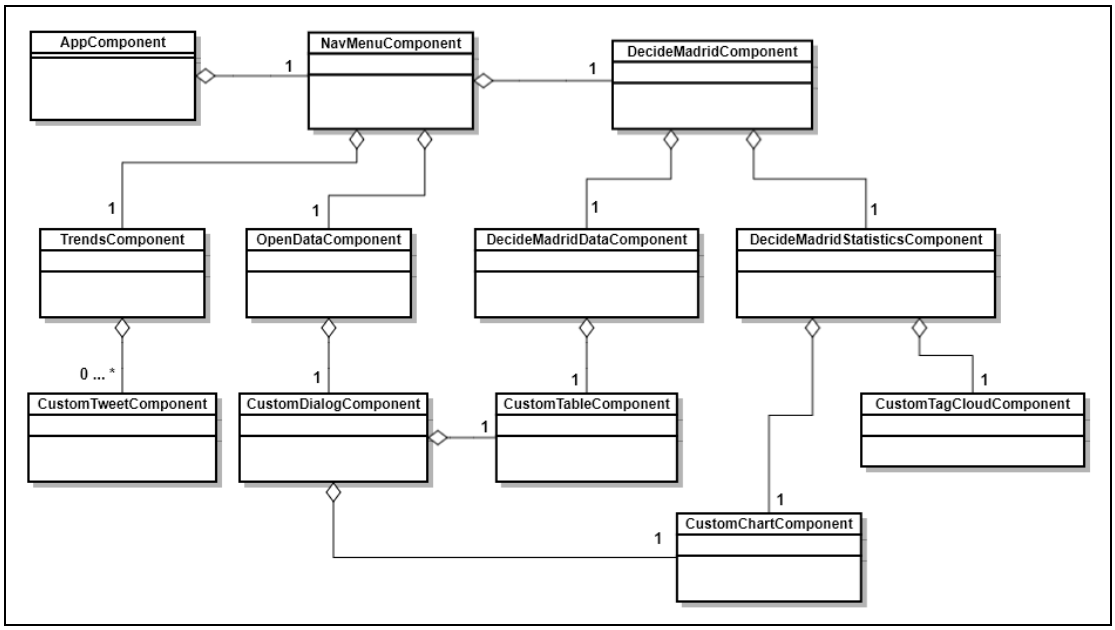

Figura 2 – Diagrama de clases de la capa del navegador

Home

Se trata del componente que se emplea para controlar la página principal de la aplicación. Una página que sirva para presentar la aplicación y dirigir al usuario a las páginas enfocadas en tratar los datos.

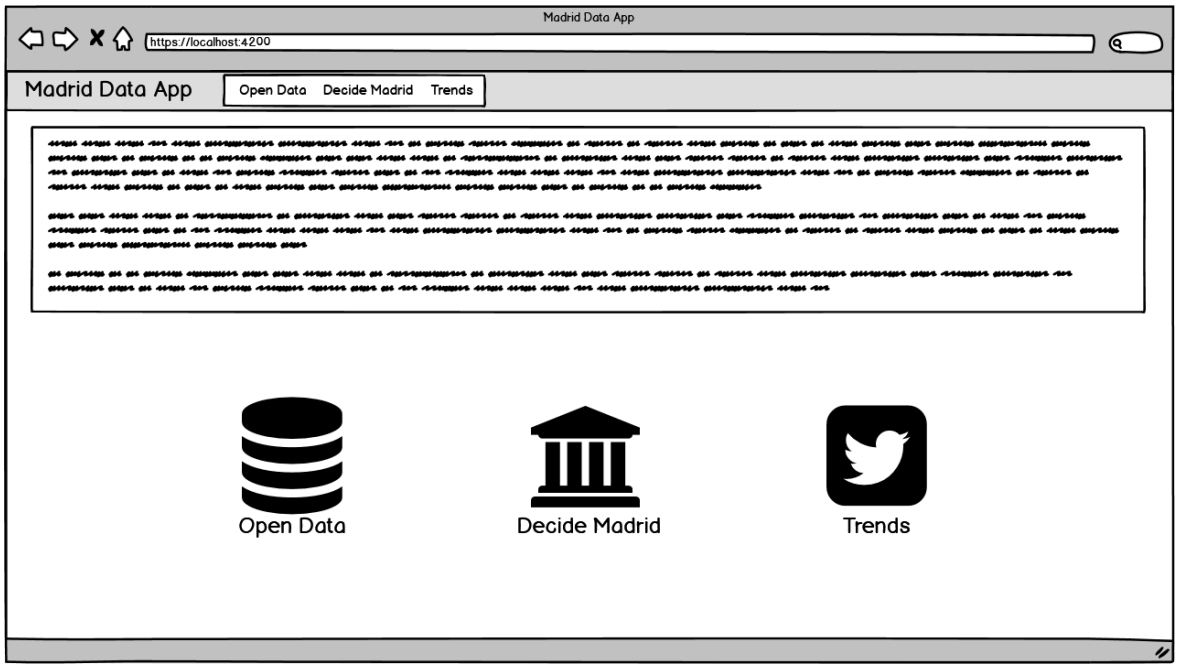

Figura 3 – Concepto de pantalla inicial

Decide-madrid

Este componente se encarga de gestionar la página que trabaja los datos extraídos de la plataforma Decide Madrid. En este componente se hace uso de los componentes decide-madrid-statistics y decide-madrid-data.

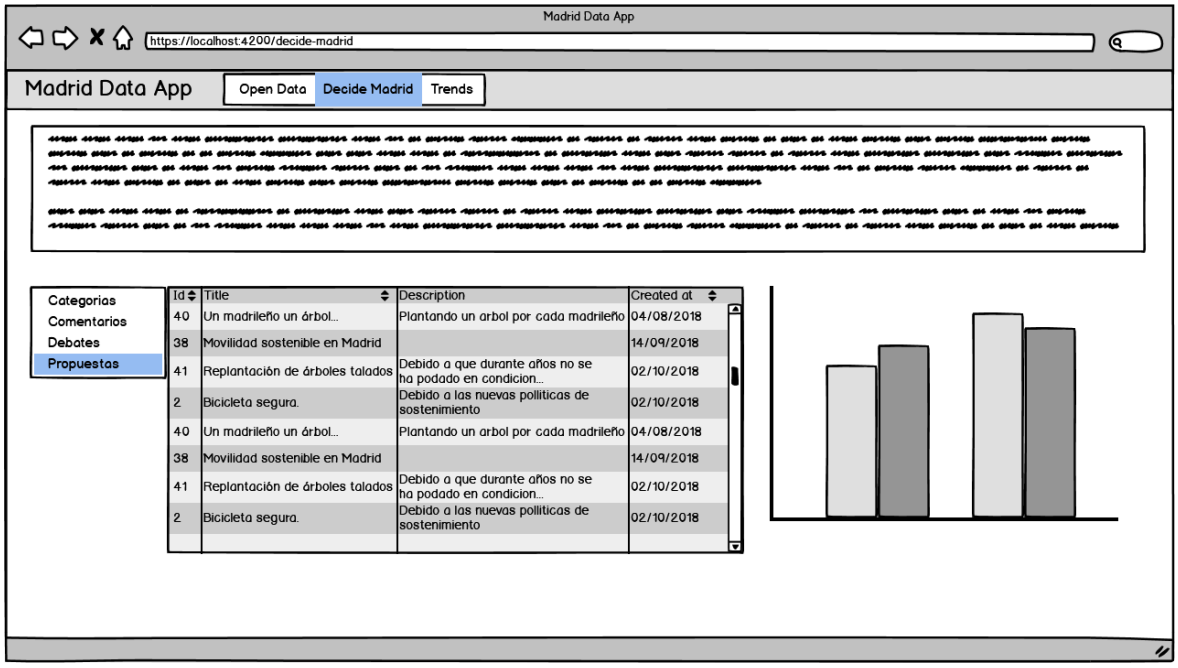

Figura 4 – Concepto inicial de pantalla de visualización de datos

Decide-madrid-data

Este componente se encarga de visualizar los datos no estadísticos de Decide Madrid. En él se hace uso del componente custom-table.

Decide-madrid-statistics

Este componente se encarga de obtener y visualizar los datos estadísticos recogidos en la plataforma Decide Madrid. Para mejorar la visibilidad de los datos estadísticos se hace uso de componentes como custom-chart o custom-tag-cloud.

Trends

Este componente se encarga de gestionar la página que trabaja con los datos extraídos de Twitter. En este componente se incluirán componentes como custom-tweet o custom-table.

Open-data

Este componente se encarga de gestionar la página que trabaja con los datos extraídos de la plataforma de datos abiertos del ayuntamiento de Madrid. En este componente se incluirán componentes como custom-table o custom-chart.

Nav-menu

Este componente está enfocado en gestionar la estructura de la aplicación web, permitiendo tener el contenido principal de la aplicación siempre visible y mostrando las opciones de menú en el lateral izquierdo de la aplicación cuando el usuario así lo desee.

Custom-chart
Este componente se encarga de mostrar gráficas de los datos que reciba del componente que lo contiene.

Custom-dialog
Este componente se encarga de mostrar otros componentes en una ventana emergente. Puede ser empleado para mostrar, por ejemplo, tablas, gráficas o simplemente mensajes que se quieran transmitir al usuario.

Custom-table
Este componente se encarga de mostrar tablas de datos. Debe de ser capaz de trabajar con cualquier tipo de dato y poder ordenarse en función del campo seleccionado.

Custom-tag-cloud
Este componente se encarga de mostrar una nube de palabras a partir de los datos que reciba del componente que lo contenga.

Custom-tweets
Este componente se encarga de visualizar el contenido de los tweets que reciba del componente que lo contenga.

Services
No se trata de un componente, pero he decidido agrupar en una misma carpeta en el proyecto todos los servicios, ya sea para comunicarse con el servidor a través del API REST o para dotar de funcionalidad común a los componentes.

3.2 Capa del servidor
Dado que la intención de la capa del servidor es la de acceder a los datos de la base de datos y suministrarlos a la capa del navegador mediante un API REST, el servidor se ha diseñado segmentado en los siguientes paquetes:

- Entities
- Services
- Controllers

Entities
En este paquete se incluyen todas las clases Java de cada una de las tablas de la base de datos, así como las clases adicionales necesarias para gestionar las claves compuestas de las tablas o para gestionar los tipos de datos relativos a las estadísticas.

Services
En este paquete se incluyen todas las clases que implementan JPA Repositories y permiten el acceso a los datos de la base de datos.

Controllers
En este paquete se localizan las clases que gestionan las peticiones HTTP y que invocan los métodos del API REST.

3.3 Capa de persistencia

En esta capa no ha habido mucho diseño, puesto que todas las bases de datos han sido suministradas por el tutor y realizadas por terceras personas. La única decisión de diseño que he tomado ha sido la de incluir todas las bases de datos en una única base de datos para evitar tener que gestionar distintas conexiones desde la capa del servidor.

4 Desarrollo

4.1 Capa del navegador

La capa del navegador está formada por los módulos *app.module*, *material.module* y *app-routing.module*. Un módulo de Angular nos permite organizar el código agrupando componentes, servicios y demás funcionalidades propias de Angular en bloques. También nos permite importar otros módulos o exportarlo para extender la funcionalidad de dicho módulo. En este caso, esta aplicación está incluida en el módulo *app.module*, definido en el fichero *app.module.ts*. Para extender su funcionalidad se han importado los módulos *material.module* (definido en el fichero *material.module.ts*) que nos permite usar la funcionalidad y los componentes propios de Angular Material, y el módulo *app-routing.module* (definido en el fichero *app-routing.module.ts*) que permite gestionar el enrutamiento de componentes para simular la navegación entre páginas.

El código que ha sido desarrollado para este TFG está incluido en el módulo *app.module* y se compone de los componentes y servicios que son detallados en los siguientes apartados.

4.1.1 Home

El componente Home está formado por los ficheros home.component.ts, home.component.html y home.component.css. Su funcionalidad consiste en ser la *landing page* de la aplicación. En su interacción con el usuario permite que el usuario navegue a las páginas del resto de la aplicación, para ello se hace uso de la funcionalidad del módulo Router de Angular. También hace uso del servicio *MenuOptionsService* para poder listar las páginas a las que el usuario puede navegar.

En la figura 11 se muestra el contenido del archivo *home.component.ts*. Este fichero contiene la implementación de la clase *HomeComponent*. Esta clase posee un método constructor vacío, pero requerido, ya que en él se hace la inyección del servicio de enrutamiento de Angular (*Router*) y del servicio *MenuOptionsService*, que contiene las opciones de menú de toda la aplicación. También se ha implementado el método *enroute*, se trata del método que se invoca desde el interfaz de usuario para realizar el enrutamiento. Por último, se ha creado un método *getter* para obtener las opciones de menú, excluyendo la pantalla que representa este componente. Los métodos *getter* en Angular tienen la propiedad de poder ser usados como si se tratase de una variable de la clase.

En la figura 12 puede observarse la estructura HTML del componente en el fichero home.component.html. El aspecto final de este componente puede verse en la figura 5.

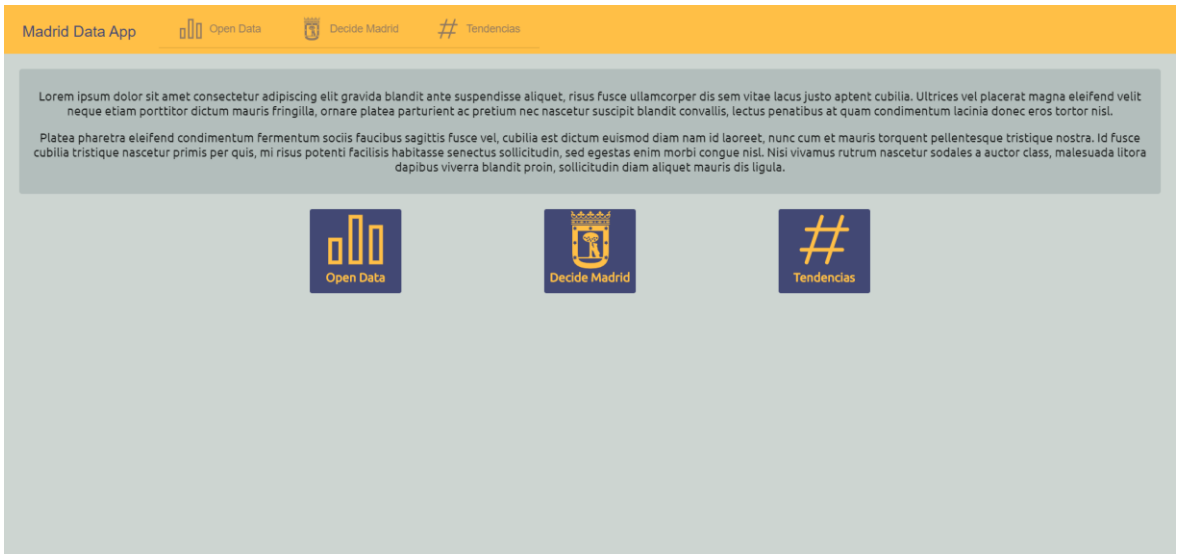

Figura 5 – Visualización componente Home

4.1.2 Decide-madrid

Este componente está formado por los ficheros decide-madrid.component.ts, decide-madrid.component.html y decide-madrid.component.css. Este componente se encarga de mostrar los datos relacionados a la plataforma Decide Madrid, por lo que hace uso de los componentes *decide-madrid-data* y *decide-madrid-statistics*. También hace uso del servicio *proposal*, para obtener los datos relacionados con las propuestas recogidas en Decide Madrid.

Para que visualmente fuera sencilla la transición entre los *decide-madrid-data* y *decide-madrid-statistics*, he empleado el componente de Angular Material *mat-tab*. Se trata de un componente que permite organizar contenido en vistas separadas donde solamente se puede ver una a la vez, además permite cambiar de vista con la pulsación de un botón y reproduce una animación mientras se cambia de vista.

En la figura 13, que muestra la estructura HTML del componente, se puede comprobar cómo se incluyen los componentes *decide-madrid-data* y *decide-madrid-statistics* mediante el uso de sus etiquetas HTML (*app-decide-madrid-data* y *app-decide-madrid-statistics*).

4.1.3 Decide-madrid-data

Este componente está formado por los ficheros decide-madrid-data.component.ts, decide-madrid-data.component.html y decide-madrid-data.component.css. Este componente muestra tablas con los datos almacenados por Decide Madrid. Para mostrar los datos en tablas se hace uso del componente custom-table.

Mediante el uso del componente de Angular Material *mat-tree* se muestran con una estructura de árbol las opciones disponibles, permitiendo organizarlas por temática principal, como puede verse en la figura 6. Al seleccionar una opción del árbol, se realiza una petición HTTP al servidor (si se trata de la primera vez, ya que durante la misma sesión se guardan los resultados obtenidos de servidor para disminuir las peticiones de datos) y cuando se reciben los datos estos se inyectan en el componente *custom-table* para que los muestre en pantalla. En la figura 14 se puede ver cómo queda la estructura HTML

y diferenciar la estructura anidada del árbol de opciones y el uso del componente *custom-table* con la etiqueta HTML *app-custom-table*.

Figura 6 – Visualización componente decide-madrid-data

4.1.4 Decide-madrid-statistics

Este componente está formado por los ficheros decide-madrid-statistics.component.ts, decide-madrid-statistics.component.html y decide-madrid-statistics.component.css. En este componente se pueden ver diferentes elementos gráficos para comprender los datos estadísticos obtenidos de Decide Madrid, para ello se hace uso de los componentes *custom-chart* y *custom-tag-cloud*.

Para facilitar la transición entre distintas gráficas y conceptos estadísticos he hecho uso del componente de Angular Material *mat-tree*. Se trata de un componente que permite mostrar un árbol jerárquico con funcionalidades de acordeón. Es el mismo componente que se ha empleado en *decide-madrid-data*. Una vez se selecciona una opción del árbol, se realiza la petición indicada al servidor y, cuando se recibe la respuesta, se inyectan los datos al componente *custom-chart* o *custom-tag-cloud*, dependiendo de la opción seleccionada. En la figura 7 se puede ver cómo queda visualmente este componente y en la figura 15 cómo se organiza la estructura HTML.

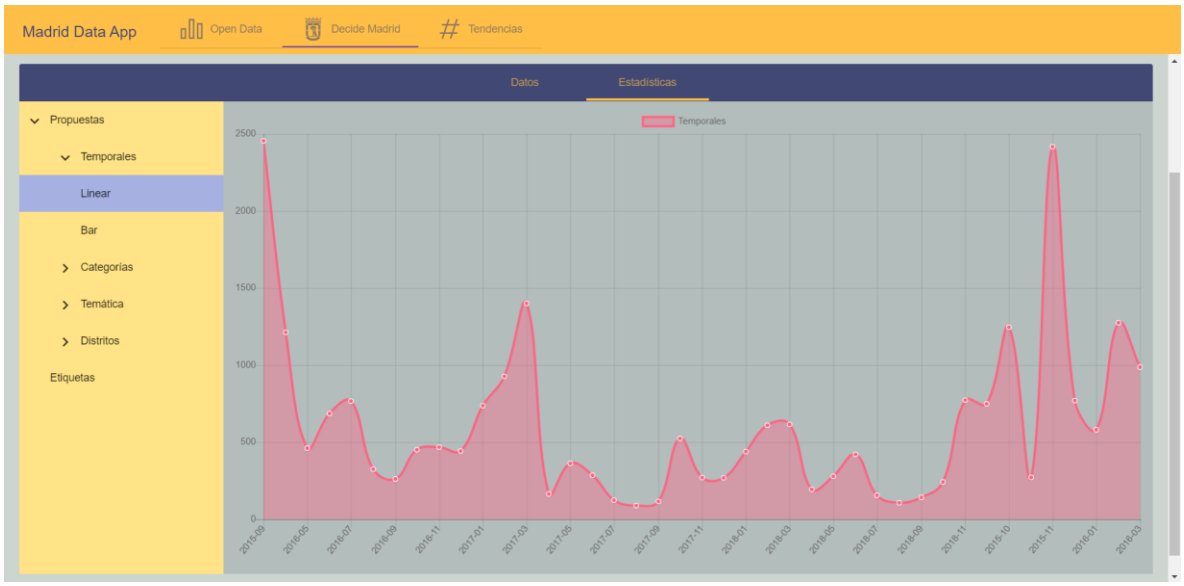

Figura 7 – Visualización componente decide-madrid-statistics

4.1.5 Trends

Este componente está formado por los ficheros trends.component.ts, trends.component.html y trends.component.css. Este componente es parte de la navegación de la aplicación y su funcionalidad consiste en visualizar datos estadísticos resultado del análisis de tweets en lo relativo a la ciudad de Madrid. Para ello se hace uso del componente custom-tweets.

4.1.6 Open-data

Este componente está formado por los ficheros open-data.component.ts, open-data.component.html y open-data.component.css. Este componente es parte de la navegación y está ideado para albergar otros contenidos de la plataforma de datos abiertos del Ayuntamiento de Madrid.

4.1.7 Nav-menu

Este componente está formado por los ficheros nav-menu.component.ts, nav-menu.component.html y nav-menu.component.css. Este componente es el que da la estructura visual principal de la aplicación, ya que contiene en él el menú superior y todos los componentes que forman parte de la navegación.

Como puede observarse en la figura 16, este componente hace uso de los componentes mat-toolbar, para el menú superior, y el componente mat-sidenav. El componente mat-sidenav se ha empleado en la construcción de la aplicación para el uso de un menú lateral en una futurible adaptación a pantallas pequeñas o de terminales móviles, este componente emplea la etiqueta HTML *mat-sidenav* para incluir en ella el contenido del menú y la etiqueta *mat-sidenav-content* para incluir en ella el contenido principal del componente. Es en el interior de esta etiqueta donde se ubica la etiqueta HTML *router-oulet*, esta etiqueta es interpretada por el motor de Angular para sustituirla por el componente al que se haga referencia con el servicio de enrutamiento. En la figura 17 aparece la clase NavMenuComponent, esta clase contiene el método *menuOptions*, que obtiene las opciones de menú.

4.1.8 Custom-chart

Este componente está formado por los ficheros custom-chart.component.ts, custom-chart.component.html y custom-chart.component.css. Este componente se encarga de mostrar una gráfica con los datos que recibe del componente que lo importa.

Para mostrar las gráficas en este componente se ha usado el módulo ng2-charts [9]. Se trata de un módulo desarrollado por terceros y que provee de directivas Angular que se aplican sobre un elemento *canvas* de la vista del componente, como puede observarse en la figura 18.

Este componente tiene dos propiedades de entrada para que el componente reciba los datos con los que generar el gráfico (`chartData`) y la configuración del mismo (`options`). Angular posee la capacidad para detectar que este tipo de propiedades sufren cambios y permite al desarrollador ejecutar código ante el evento que emite el cambio mediante el método ngOnChanges, como puede verse en la figura 19. En este componente el método ngOnChanges sobre escribe las variables empleadas por las directivas del módulo ng2-charts.

4.1.9 Custom-dialog

Este componente está formado por los ficheros custom-dialog.component.ts, custom-dialog.component.html, custom-dialog.component.css y custom-dialog.service.ts. Este componente gestiona el contenido que debe mostrarse en la ventana emergente que se abre desde su servicio.

Este componente hace uso del módulo MatDialogModule, que provee de un servicio para gestionar la interacción con el diálogo y directivas para adecuar el contenido del mismo.

Debido a las características de este componente de Angular Material, se ha optado por crear un servicio específico para evitar tener que implementar el uso del servicio que provee MatDialogModule en cada componente que se desee usar el diálogo. Además, este servicio construye el modelo de datos adecuado en función de los parámetros del método que permite abrirlo como puede verse en la figura 20.

Como este componente permite ver tablas, gráficas o cualquier tipo de contenido, se hace uso en él de los componentes custom-table y custom-chart. En la figura 21 puede verse la estructura HTML del componente y el uso de estos componentes.

4.1.10 Custom-table

Este componente está formado por los ficheros custom-table.component.ts, custom-table.component.html y custom-table.component.css. En este componente se emplea el componente de Angular Material mat-table, que permite visualizar filas de datos en formato de tabla, ordenar las filas de datos dependiendo de la columna que se seleccione (haciendo uso del componente mat-sort-header) o paginar los datos mostrados (haciendo uso del componente mat-paginator) como puede verse en la figura 22.

En este componente se trabaja con todo tipo de modelo de dato, ya que Angular permite formatear los valores en la vista html. Para poder visualizar la tabla correctamente, se ha optado por ocultar las columnas de tipo string en la que alguno de sus valores tenga una longitud superior a 60 caracteres. Para poder ver estos datos que se omiten en la tabla, se debe hacer click en la fila de la tabla de la que se quiera ver esta información, al hacerlo, se

muestra una fila en la tabla que contiene todos los campos omitidos, esto puede observarse en la figura 8.

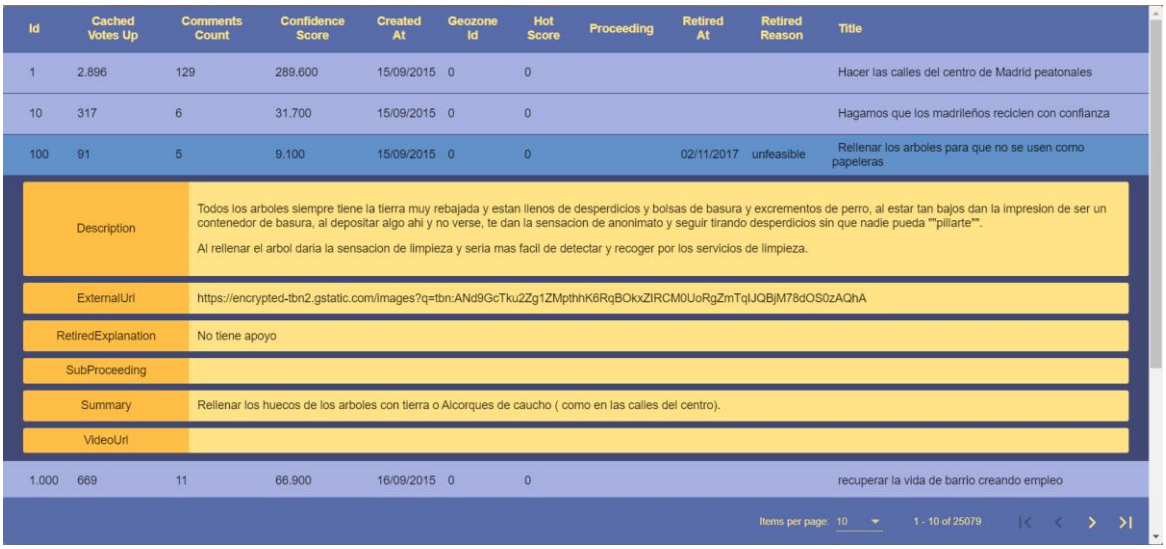

Figura 8 – Fila extendida con datos omitidos en la tabla

4.1.11 Custom-tag-cloud

Este componente está formado por los ficheros custom-tag-cloud.componente.ts, custom-tag-cloud.componente.html y custom-tag-cloud.componente.css. En este componente se emplea el módulo *angular-tag-cloud-module* [6], con él se puede generar una nube de palabras haciendo uso de su componente angular-tag-cloud.

En la figura 23 que contiene la estructura HTML del componente y en ella puede verse el empleo del componente de nube de palabras mediante la etiqueta angular-tag-cloud. Con la figura 9 puede verse el resultado final de su uso en la sección de estadísticas de la pantalla Decide Madrid.

Figura 9 – Ejemplo de uso del componente custom-tag-cloud

4.1.12 Custom-tweets

Este componente está formado por los ficheros custom-tweets.component.ts, custom-tweets.component.html y custom-tweets.component.css. Para renderizar los tweets se hace uso del componente ngx-tweet incluido en el módulo NgxTweetModule [7]. El componente ngx-tweet renderiza, con la estética de Twitter, un tweet recibiendo como parámetro el identificador del tweet.

En la figura 24 puede verse el modo de empleo del componente NgxTweet y en la figura 10 puede verse el resultado final.

Figura 10 – Ejemplo de uso del componente custom-tweets

4.1.13 Services

Entre los servicios incluidos en la carpeta services se encuentran dos tipos de servicios.

Un tipo de servicio consiste en un servicio para compartir datos, como por ejemplo el servicio menú-options.service.ts, que contiene las opciones del menú de navegación de la aplicación.

El otro tipo de servicio consiste en servicios de obtención de datos del servidor mediante el API REST. Existe un servicio de este tipo por cada controller existente en el servidor. Hacen uso del módulo HttpClientModule de Angular, que provee de un servicio de gestión de peticiones HTTP.

La figura 25 contiene el código del servicio ProposalService, este servicio es ligeramente más completo que el resto de los servicios del API REST. Como puede observarse en el método GetProposals de este servicio, se hace uso de los métodos publishReplay y refCount, junto con la variable del servicio *cachedProposals*, para realizar la petición al servidor solo la primera vez.

4.2 Capa del servidor

4.2.1 Entities

Este package de Java es fruto del uso de la herramienta de ingeniería inversa de la perspectiva JPA de Eclipse. Con esta herramienta se pueden generar las clases Java de las entidades de la base de datos de manera automática. Como consecuencia del uso de esta herramienta y dado que algunas tablas de la base de datos tienen claves primarias compuestas, se ha generado una clase Java por cada clave compuesta para representarlas en las entidades. Su nomenclatura está formada por el nombre de la entidad de la tabla seguido de los caracteres *PK*.

4.2.2 Services

Este package de Java contiene una interfaz por cada entidad principal del package *entities*. Cada una de estas interfaces extiende la interfaz JpaRepository, que serán autocompletadas por Spring al usar la anotación `@Repository`.

4.2.3 Controllers

En este package se agrupan las clases Java que implementan los controladores del API REST. Se ha creado una clase por cada una de las entidades principales del package *entities*. Estas clases hacen uso de los repositorios alojados en el package *repositories* para obtener los datos de la base de datos. Para simplificar el desarrollo de estas clases se han utilizado varias anotaciones de *Spring*.

La anotación `@CrossOrigin` indica al controlador que permita el intercambio de recursos entre distintos dominios. Esta anotación requiere como parámetro que se le indique la ruta de los orígenes de datos a los que se desea permitir el cruce de datos. He necesitado usar esta anotación debido a la configuración del entorno de desarrollo, ya que la aplicación servidor se ejecuta en localhost:8080 y la aplicación cliente se ejecuta en el puerto 4200 de localhost.

La anotación `@RequestController` indica que esa clase es un controlador del API REST.

Existe la posibilidad de englobar las rutas de un controlador bajo una misma URL, para ello se hace uso de la anotación `@RequestMapping`. Esta anotación recibe como parámetro el texto de la ruta que se desea hacer común.

Para poder inyectar un repositorio en un controlador se hace uso de la anotación `@Autowired`. Esta anotación vincula el Bean generado a partir del repositorio con el Bean generado a partir del controlador.

Por último, se ha hecho uso de la anotación `@GetMapping`. Esta anotación indica que el método que le sigue responde a la petición HTTP GET indicada como parámetro de la anotación. Si se usa junto a la anotación `@RequestMapping`, la ruta usada en el navegador debe ser una composición de la ruta de `@RequestMapping` seguida de la ruta de `@GetMapping`. En este proyecto no se ha dado el caso de necesitar otro tipo de petición HTTP pero, de haberse necesitado, existen distintas anotaciones para cada método HTTP, como por ejemplo `@PostMapping`.

En las figuras 26 y 27 pueden verse las clases ProposalRepository y ProposalController, como ejemplo de la implementación de la capa del servidor.

4.3 Capa de persistencia

La capa de persistencia no ha requerido desarrollo más allá de las modificaciones necesarias para que la base de datos pudiera ser usada junto a Java Spring y JPA.

Entre los cambios necesarios se encuentra la creación de claves primarias para todas las tablas. Algunas tablas poseen campos que por sí mismos han podido ser utilizados como Primary Key. En cambio, otras tablas han requerido del uso de claves compuestas, alguna clave compuesta ha tenido que usar todos los campos de la tabla, o la creación de un campo numérico con autoincremento, ya que ni con el uso de todos sus campos es posible distinguir todas las entradas de la tabla.

5 Integración, pruebas y resultados

Las pruebas realizadas al software desarrollado consisten, principalmente, en la comprobación de que los datos obtenidos en origen (base de datos) y los mostrados en destino (interfaz web) son los mismos. Este tipo de pruebas se han realizado en el entorno desarrollo, compuesto por la instancia SQL (corriendo en localhost:3306), la aplicación de Java Spring (corriendo en localhost:8080) y la aplicación Angular (corriendo en localhost:4200). Estas pruebas han requerido de la integración de las tres capas del proyecto para su aplicación siendo realizadas mediante el navegador web realizando peticiones al API REST desarrollado y comprobando que los resultados obtenidos coincidían con los datos de la base de datos.

En lo referente a las pruebas de funcionalidad de los componentes de Angular, se han testeado los componentes de diferentes maneras. Los componentes miembros de la navegación de la aplicación (home, open-data, decide-madrid, tren y nav-menu) han sido probados mediante la navegación por la aplicación, forzando distintas situaciones. Los componentes decide-madrid-data y decide-madrid-statistics han sido los componentes principales en las pruebas de obtención de datos, comprobando que los datos recibidos coinciden con los datos en origen y que la selección marcada en el listado opciones, tanto para los datos a mostrar en tabla como para los distintos tipos de gráficas, provoca cambios en los datos mostrados por los componentes custom-chart, custom-table y custom-tag-cloud. Por otra parte, el componente custom-chart se ha probado, en un primer momento, con un conjunto de datos preparados; posteriormente se ha testeado con los datos obtenidos de la base de datos, comprobando que los diferentes tipos de gráficas trabajan correctamente con los datos suministrados. Pruebas similares se han realizado en el componente custom-table, en un primer momento se comprobó la funcionalidad inicial con un conjunto de datos preparados, tras comprobar su correcto funcionamiento se probó el componente con los datos obtenidos de la base de datos; se ha comprobado la funcionalidad de ordenación de los resultados y la paginación de los elementos de la tabla, así como la funcionalidad de ocultar columnas con datos de gran tamaño. Las pruebas aplicadas al componente custom-tag-cloud han sido meramente visuales, se ha comprobado que las palabras más grandes generadas por el componente de terceros empleado coinciden con las palabras más repetidas que recibe el componente. El componente custom-dialog se ha probado junto a los componentes custom-table y custom-chart, comprobando que se muestra el componente indicado en una acción lanzada por un botón de la interfaz. El componente custom-tweets se ha probado visualmente, contrastando tweets visualizados en la plataforma Twitter contra los mostrados por el componente de terceros empleado en custom-tweets.

Los datos empleados han sido obtenidos de la base de datos de Decide Madrid. Esta base de datos contiene los datos almacenados sobre un total de 25.079 propuestas realizadas, vinculadas a 129 barrios de la ciudad de Madrid, repartidos en 21 distritos. También contiene 121.875 comentarios realizados por los usuarios y 3.628.524 de votos registrados. Las propuestas están categorizadas en 30 categorías y existe una variedad de 3820 etiquetas asignadas a las propuestas.

Para evaluar la usabilidad de la aplicación se propone el uso de una combinación de sesiones de Thinking Aloud [17] y formularios USE [18] con distintos usuarios. Las

sesiones de Thinking Aloud son realizadas por una persona que va a probar la aplicación (usuario) y un evaluador. Durante la sesión, el usuario debe completar una serie de tareas preparadas por el evaluador y comentar en voz alta sus impresiones. Mientras, el evaluador debe tomar anotaciones sobre los comentarios del usuario, tomar mediciones de tiempo y verificar que se completan las tareas, procurando no interactuar con el usuario. Una vez finalizada la sesión de Thinking Aloud, el usuario deberá responder las preguntas de un formulario USE con valoración de escala tipo Likert [19].

6 Conclusiones y trabajo futuro

6.1 Conclusiones

La publicación de datos abiertos está creciendo constantemente. Tanto la población, como investigadores reclaman más datos abiertos. Su enfoque sigue siendo el de ser empleados en tareas automatizadas, lo que dificulta su interpretación. Además, el empleo de aplicaciones web permite al usuario un rápido acceso a la información, sin la necesidad de la instalación de software específico, y permite al desarrollador mejorar la herramienta sin obligar al usuario a actualizarla. Es por todo esto que el proyecto desarrollado puede llegar a tener cierta importancia.

Durante el desarrollo de este proyecto he comprobado que las aplicaciones web pueden llegar a ser muy potentes y versátiles. Frameworks como Angular o React permiten, al desarrollador, crear herramientas complejas, pero visualmente sencillas, y frameworks como .NETCore y Spring simplifican el trabajo de los desarrolladores.

Por otra parte, la cantidad de datos abiertos es abrumadora y su análisis se antoja complicado, aunque necesario. El estudio de estos datos puede hacer que la tecnología se perfeccione y mejore la vida de la gente.

6.2 Trabajo futuro

Este proyecto solo abarca una cantidad muy reducida de los datos publicados por el portal de datos abiertos del Ayuntamiento de Madrid. El primer paso a dar, en un futuro, debería de ser el de añadir más fuentes de datos a este proyecto. Cuantos más datos se puedan consultar en la aplicación desarrollada, más valor tendrá.

Otro aspecto a mejorar del proyecto puede consistir en aumentar la variedad de recursos visuales de la aplicación, como el uso de mapas de posicionamiento geográfico o la creación de pantallas mixtas, en las que se puedan visualizar gráficas y cartografías simultáneamente.

Referencias

[1] "¿Qué son los datos abiertos?", Open Knowledg Foundation https://opendatahandbook.org/guide/es/what-is-open-data/.
[2] "Portal de datos abiertos del Ayuntamiento de Madrid", Ayuntamiento de Madrid https://datos.madrid.es/
[3] Ajitesh Shukla, "Building Web Apps with Spring 5 and Angular", Packt, 2017
[4] "Angular", Google https://angular.io
[5] "Angular Material", Google https://material.angular.io
[6] "angular-tag-cloud-module", Danny Koppenhagen https://github.com/d-koppenhagen/angular-tag-cloud-module
[7] "ngx-tweet", adrael https://www.npmjs.com/package/ngx-tweet
[8] "Angular Flex Layout", https://github.com/angular/flex-layout/wiki/Responsive-API
[9] "ng2-charts", https://valor-software.com/ng2-charts/
[10] "React", Facebook Open Source https://es.reactjs.org/
[11] "Vue", https://vuejs.org/
[12] "Estándar componente web", https://html.spec.whatwg.org/multipage/custom-elements.html
[13] "LAMP", Wikipedia https://es.wikipedia.org/wiki/LAMP
[14] "Microsoft .NET", Microsoft https://dotnet.microsoft.com/
[15] "Java EE at a Glance", Oracle https://www.oracle.com/es/java/technologies/java-ee-glance.html
[16] "Spring", https://spring.io/
[17] "Think aloud protocol", Wikipedia https://en.wikipedia.org/wiki/Think_aloud_protocol
[18] A. M. Lund, "Measuring usability with the USE questionnaire," STC Usability SIG Newsletter, vol. 8, no. 2, 2001.
[19] "Escala Likert", Wikipedia https://es.wikipedia.org/wiki/Escala_Likert

Glosario

API	Application Programming Interface.
CSV	Comma-Separated Values.
Framework	Esquema o estructura que se establece y que se aprovecha para desarrollar y organizar un software determinado.
HTML	HyperText Markup Language
Landing page	Pantalla que se muestra al cargar la aplicación por primera vez.
REST	Representational state transfer.
SAMUR	Servicio de Asistencia Municipal de Urgencia y Rescate.
XLSX	Microsoft Excel Open XML Spreadsheet.

Anexos

A Figuras

```typescript
import { Component } from '@angular/core';
import { Router } from '@angular/router';

import { MenuOptionsService } from '../services/menu-options.service';

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
export class HomeComponent {
  constructor(private router: Router, public menuOptionsService: MenuOptionsService) {
  }

  enroute(route: string) {
    this.router.navigate([route]);
  }

  get menuOptions(): Models.MenuOption[] {
    return this.menuOptionsService && this.menuOptionsService.options ?
      this.menuOptionsService.options.filter(x => x.title.toLowerCase() != 'home') : [];
  }
}
```

Figura 11 – home.component.ts

```html
<section class="section">
  <div class="section-content">
    <div fxLayout="row" class="intro intro-text">
      <span [innerHTML]="introText"></span>
    </div>

    <div fxLayout="row">
      <div fxFlex="20" fxHide.xs></div>
      <div fxFlex="60" [fxFlex.xs]="100">
        <div fxLayout.xs="column" [fxFlex.gt-xs]="100/menuOptions.length" *ngFor="let opt of menuOptions">
          <div fxLayout="column" class="home-option">
            <div fxLayout="column" class="home-option-item" (click)="enroute(opt.route)">
              <mat-icon fxFlex="100" class="home-option-icon" svgIcon="{{opt.iconSecondary}}"></mat-icon>
              <span fxFlex="100" class="home-option-title">{{opt.title}}</span>
            </div>
          </div>
          <div fxHide.gt-xs style="height: 15px;"></div>
        </div>
      </div>
      <div fxFlex="20" fxHide.xs></div>
    </div>
  </div>
</section>
```

Figura 12 - home.component.html

```
<section class="section">
  <div class="section-content">
    <div fxLayout="row" class="intro intro-text">
      <span [innerHTML]="introText"></span>
    </div>

    <div fxLayout="row" class="M20">
      <mat-tab-group class="tabs-group" mat-align-tabs="center" dynamicHeight="true" (selectedIndexChange)="currentTab = $event">
        <mat-tab label="Datos">
          <div class="tab-container">
            <app-decide-madrid-data *ngIf="currentTab == 0"></app-decide-madrid-data>
          </div>
        </mat-tab>
        <mat-tab label="Estadísticas">
          <div class="tab-container">
            <app-decide-madrid-statistics *ngIf="currentTab == 1"></app-decide-madrid-statistics>
          </div>
        </mat-tab>
      </mat-tab-group>
    </div>
  </div>
</section>
```

Figura 13 – decide-madrid-component.html

```
<section class="MINH45VH">
  <div class="inherit-min-h" fxLayout="row">
    <div class="tree-options" fxFlex="18">
      <mat-tree [dataSource]="treeOptions" [treeControl]="treeControl">
        <mat-tree-node (click)="selectNode(node)"
        [ngClass]="{'selected-node': node.id == selectedNode?.id, 'tree-node': node.id != selectedNode?.id, 'flat-tree': node.level == 0 }"
        *matTreeNodeDef="let node"
        matTreeNodePadding>
          <span [ngClass]="{'PL20': node.level == 0}">{{node.name}}</span>
        </mat-tree-node>
        <mat-tree-node class="tree-node" (click)="toggleNode(node)" *matTreeNodeDef="let node; when: hasChild" matTreeNodePadding>
          <button mat-icon-button>
            <mat-icon class="mat-icon-rtl-mirror">
              {{treeControl.isExpanded(node) ? 'expand_more' : 'chevron_right'}}
            </mat-icon>
          </button>
          {{node.name}}
        </mat-tree-node>
      </mat-tree>
    </div>
    <div class="content-container" fxFlex="82">
      <div class="loading-content" *ngIf="isLoadingData == true">
        <mat-progress-spinner class="example-margin" [mode]="'indeterminate'" [value]="60"></mat-progress-spinner>
      </div>
      <app-custom-table *ngIf="tableData?.length > 0 && isLoadingData == false" [inputData]="tableData" [options]="tableOptions">
      </app-custom-table>
    </div>
  </div>
</section>
```

Figura 14 – decide-madrid-data.component.html

```
<section class="MINH45VH">
  <div class="inherit-min-h" fxLayout="row">
    <div class="tree-options" fxFlex="18">
      <mat-tree [dataSource]="dataSource" [treeControl]="treeControl">
        <mat-tree-node (click)="selectNode(node)"
        [ngClass]="{'selected-node': node.id == selectedNode?.id, 'tree-node': node.id != selectedNode?.id}"
        *matTreeNodeDef="let node"
        matTreeNodePadding>
            <span>{{node.name}}</span>
        </mat-tree-node>
        <mat-tree-node class="tree-node" (click)="toggleNode(node)" *matTreeNodeDef="let node; when: hasChild" matTreeNodePadding>
          <button mat-icon-button>
            <mat-icon class="mat-icon-rtl-mirror">
              {{treeControl.isExpanded(node) ? 'expand_more' : 'chevron_right'}}
            </mat-icon>
          </button>
          {{node.name}}
        </mat-tree-node>
      </mat-tree>
    </div>
    <div fxFlex="82">
      <div class="loading-content" *ngIf="isLoadingData == true">
        <mat-progress-spinner class="example-margin" [mode]="'indeterminate'" [value]="60"></mat-progress-spinner>
      </div>
      <app-custom-tag-cloud *ngIf="showComponent == 'cloud'" [inputData]="dataStatistics"></app-custom-tag-cloud>
      <app-custom-chart *ngIf="showComponent == 'chart'" [chartData]="chart?.data" [options]="chart?.options"></app-custom-chart>
    </div>
  </div>
</section>
```

Figura 15 – decide-madrid-statistics-component.html

```
<div class="container" [class.is-mobile]="mobileQuery.matches">
  <mat-toolbar class="toolbar">
    <button mat-icon-button (click)="snav.toggle()" *ngIf="mobileQuery.matches"><mat-icon>menu</mat-icon></button>
    <h1 [routerLink]="['/home']" class="app-name">{{appName}}</h1>
    <div class="ML30" *ngIf="!mobileQuery.matches">
      <nav mat-tab-nav-bar>
        <a mat-tab-link *ngFor="let opt of menuOptions" class="menu-link" [active]="activeLink == opt.route" routerLink="{{opt.route}}">
          <mat-icon class="menu-icon" svgIcon="{{opt.iconPrimary}}"></mat-icon>{{opt.title}}
        </a>
      </nav>
    </div>
  </mat-toolbar>

  <mat-sidenav-container class="sidenav-container" [style.marginTop.px]="mobileQuery.matches ? 56 : 0">
    <mat-sidenav class="sidenav" #snav [mode]="mobileQuery.matches ? 'over' : 'side'" [fixedInViewport]="mobileQuery.matches" fixedTopGap="56">
      <mat-nav-list>
        <a mat-list-item class="sidenav-item" (click)="snav.toggle()" routerLink="{{nav.route}}" *ngFor="let nav of menuOptions">
          <!-- <mat-icon class="sidenav-icon">{{nav.icon}}</mat-icon> -->
          <mat-icon class="sidenav-icon" svgIcon="{{nav.icon}}"></mat-icon>{{nav.title}}
        </a>
      </mat-nav-list>
    </mat-sidenav>

    <mat-sidenav-content>
      <router-outlet></router-outlet>
    </mat-sidenav-content>
  </mat-sidenav-container>
</div>
```

Figura 16 – nav-menu.component.html

```typescript
export class NavMenuComponent implements OnDestroy {
  @Input() appName: string;
  mobileQuery: MediaQueryList;
  public activeLink: string;

  private _mobileQueryListener: () => void;

  constructor(changeDetectorRef: ChangeDetectorRef, media: MediaMatcher,
    public menuOptionsService: MenuOptionsService, private router: Router) {

    router.events.subscribe(e => {
      if(e instanceof NavigationStart)
        this.activeLink = '';

      if(e instanceof NavigationEnd)
        this.activeLink = e.url;
    });

    this.mobileQuery = media.matchMedia('(max-width: 600px)');
    this._mobileQueryListener = () => changeDetectorRef.detectChanges();
    this.mobileQuery.addListener(this._mobileQueryListener);
  }

  ngOnDestroy(): void {
    this.mobileQuery.removeListener(this._mobileQueryListener);
  }

  get menuOptions(): Models.MenuOption[] {
    return this.menuOptionsService && this.menuOptionsService.options ? this.menuOptionsService.options : [];
  }
}
```

Figura 17 – nav-menu.component.ts

```html
<section class="chart-content">
  <div class="chart-header" *ngIf="options?.showHeader">
    <h2 class="chart-title">{{options.title}}</h2>
    <mat-form-field class="chart-selector">
      <mat-label>Chart Type</mat-label>
      <mat-select [(value)]="chartType" (selectionChange)="typeChanges()">
        <mat-option *ngFor="let type of options?.chartTypeList" [value]="type.invariant">
          {{type.name}}
        </mat-option>
      </mat-select>
    </mat-form-field>
  </div>

  <div class="chart-block" *ngIf="chartReady == true">
    <canvas baseChart
            [datasets]="data"
            [labels]="labels"
            [chartType]="chartType"
            [options]="chartOptions"
            [legend]="true">
    </canvas>
  </div>
</section>
```

Figura 18 – custom-chart.component.html

```
26    ngOnChanges(changes: SimpleChanges){
27      if(this.hexColors.length == 0){
28        for(let i = 0; i < this.options.labels.length; i++)
29          this.hexColors.push('#'+(Math.random()*0xFFFFFF<<0).toString(16));
30      }
31      this.currentOptions = changes["options"].currentValue;
32      this.data = changes["chartData"].currentValue;
33
34      this.buildChart();
35
36      this.typeChanges();
37    }
```

Figura 19 – Método ngOnChanges del componente custom-chart

```
1   import { Injectable } from '@angular/core';
2   import { MatDialogConfig, MatDialog } from '@angular/material';
3   import { CustomDialogComponent } from './custom-dialog.component';
4
5   @Injectable({
6     providedIn: 'root'
7   })
8   export class CustomDialogService {
9
10    constructor(private dialog: MatDialog) { }
11
12    openDialog(dialogType: Models.ComponentType, datasource: any, options?: any) {
13      var dialogDatasource = null;
14
15      switch(dialogType){
16        case Models.ComponentType.Table:
17          dialogDatasource = { data: datasource, options: options };
18          break;
19        case Models.ComponentType.Chart:
20          dialogDatasource = { data: datasource, options: options } as Models.ChartModel;
21          break;
22        default:
23          dialogDatasource = "NO DATA TO SHOW";
24          break;
25      }
26
27      var dialogData: Models.DialogData = { type: dialogType, datasource: dialogDatasource };
28      var config: MatDialogConfig = { minWidth: "fit-content", data: dialogData };
29
30      if(dialogData.type == Models.ComponentType.Chart)
31        config.minWidth = "60vw";
32      else if (dialogData.type == Models.ComponentType.Table)
33        config.maxHeight = "85vh";
34
35      this.dialog.open(CustomDialogComponent, config);
36    }
37  }
```

Figura 20 – Servicio del componente custom-dialog

```
<mat-dialog-content class="MH85VH MW90VW">
    <app-custom-table *ngIf="componentType == 1" [inputData]="componentData.data" [options]="componentData.options"></app-custom-table>

    <app-custom-chart *ngIf="componentType == 2" [chartData]="componentData.data" [options]="componentData.options"></app-custom-chart>
</mat-dialog-content>
```

Figura 21 – custom-dialog.component.html

```
<section class="table-section">
  <div class="table-content">
    <table class="custom-table" mat-table matSort [dataSource]="dataSource" multiTemplateDataRows>
      <!-- All Columns -->
      <ng-container *ngFor="let displayCol of displayedColumns">
        <ng-container matColumnDef="{{displayCol}}">
          <th mat-header-cell class="table-header-row" [ngClass]="{'primary': theme == 'primary','secondary': theme == 'secondary'}" *matHeaderCellDef mat-sort-header>
            <span class="capitalize PLR7"> {{formatColName(displayCol)}} </span>
          </th>
          <td mat-cell style="color: inherit;" [ngClass]="{'primary': theme == 'primary','secondary': theme == 'secondary'}" *matCellDef="let element">
            <span class="PLR7" *ngIf="formatRowValue(element[displayCol]) == 'number'"> {{ element[displayCol] | number:'1.0-3':'es' }} </span>
            <span class="PLR7" *ngIf="formatRowValue(element[displayCol]) == 'date'"> {{ element[displayCol] | date:'dd/MM/yyyy' }} </span>
            <span class="PLR7" *ngIf="formatRowValue(element[displayCol]) == 'string'"> {{ element[displayCol] }} </span>
          </td>
        </ng-container>
      </ng-container>
      <!-- Expanded Content Column - The detail row is made up of this one column that spans across all columns -->
      <ng-container matColumnDef="expandedDetail">
        <td mat-cell *matCellDef="let element" [attr.colspan]="displayedColumns.length">
          <div class="example-element-detail" [@detailExpand]="element == expandedElement ? 'expanded' : 'collapsed'"
               [@showDetail]="element == expandedElement ? 'expanded' : 'collapsed'">
            <div class="example-element-diagram" *ngFor="let expProp of expandableProp">
              <label class="example-element-position"><span class="property-label">{{expProp}}</span></label>
              <div class="example-element-symbol" [innerHtml]="element[expProp]"></div>
            </div>
          </div>
        </td>
      </ng-container>

      <tr mat-header-row [ngClass]="{'primary': theme == 'primary','secondary': theme == 'secondary'}" *matHeaderRowDef="displayedColumns; sticky: true"></tr>
      <tr mat-row class="table-definition-row example-element-row" *matRowDef="let element; columns: displayedColumns;"
        [ngClass]="{'primary': theme == 'primary','secondary': theme == 'secondary', 'expandable': expandableProp?.length > 0}"
        [class.example-expanded-row]="expandedElement === element" (click)="toggleExpandedElement(element)">
      </tr>
      <tr mat-row class="example-detail-row" [ngClass]="{'primary': theme == 'primary','secondary': theme == 'secondary'}" *matRowDef="let row; columns: ['expandedDetail']"></tr>
    </table>
  </div>
  <mat-paginator class="mat-paginator-sticky" [ngClass]="{'primary': theme == 'primary','secondary': theme == 'secondary'}"
    [pageSizeOptions]="tableOptions?.pageSizeOptions" showFirstLastButtons></mat-paginator>
</section>
```

Figura 22 – custom-table.component.html

```
<section class="cloud-section">
  <angular-tag-cloud *ngIf="dataReady" [config]="options" [data]="cloudData"></angular-tag-cloud>
</section>
```

Figura 23 – custom-tag-cloud.component.html

```
<section style="overflow-y: auto">
  <ng-container *ngFor="let tweet of tweetIds">
    <ngx-tweet tweetId="{{tweet}}"></ngx-tweet>
  </ng-container>
</section>
```

Figura 24 – custom-tweets.component.html

```typescript
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { publishReplay, refCount } from 'rxjs/operators';

@Injectable({
  providedIn: 'root'
})
export class ProposalService {
  private cachedStatistics: Observable<Models.DataStatistics[]>[] = [];
  private cachedProposals: Observable<any[]>;

  constructor(private http: HttpClient) { }

  public GetProposals(): Observable<any[]> {
    if(!this.cachedProposals){
      this.cachedProposals = this.http.get<any[]>(`${Models.WebApi.Url}/api/proposals`)
      .pipe(publishReplay(1), refCount());
    }

    return this.cachedProposals;
  }

  public GetProposalById(proposalId: number): Observable<any[]> {
    return this.http.get<any[]>(`${Models.WebApi.Url}/api/proposals/${proposalId}`);
  }

  public GetProposalStatistics(type: Models.StatisticType): Observable<Models.DataStatistics[]> {
    if(!this.cachedStatistics[type]){
      this.cachedStatistics[type] = this.http.get<Models.DataStatistics[]>(`${Models.WebApi.Url}/api/proposals/statistics/${type}`)
      .pipe(publishReplay(1), refCount());
    }

    return this.cachedStatistics[type];
  }
}
```

Figura 25 – proposal.service.ts

```java
package app.repositories;

import app.entities.Proposal;
import app.entities.ProposalStatistic;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;

@Repository
public interface ProposalRepository extends JpaRepository<Proposal, String> {
    @Query("SELECT new app.entities.ProposalStatistic(DATE_FORMAT(createdAt,'%Y-%m'), COUNT(*)) \r\n" +
            "FROM Proposal \r\n" +
            "GROUP BY DATE_FORMAT(createdAt,'%Y-%m')")
        List<ProposalStatistic>getNumProposalsMonthYear();

}
```

Figura 26 – Repositorio JPA para obtención de datos de Propuestas

```java
@CrossOrigin(origins = "http://localhost:4200")
@RestController
@RequestMapping("/api")
public class ProposalController {
    @Autowired
    private ProposalRepository proposalRepository;

    @Autowired
    private ProposalCategoryRepository proposalCategoryRepository;

    @Autowired
    private ProposalTopicRepository proposalTopicRepository;

    @Autowired
    private ProposalLocationRepository proposalLocationRepository;

    @Autowired
    private ProposalTagRepository proposalTagRepository;

    @GetMapping("/proposals")
    public List<Proposal> GetAllItems() {
        return this.proposalRepository.findAll();
    }

    @GetMapping("/proposals/statistics/time")
    public List<ProposalStatistic> GetNumProposalsMonthYear() {
        return this.proposalRepository.getNumProposalsMonthYear();
    }

    @GetMapping("/proposals/statistics/categories")
    public List<ProposalStatistic> GetNumProposalsCategories() {
        return this.proposalCategoryRepository.getNumProposalsCategories();
    }

    @GetMapping("/proposals/statistics/topics")
    public List<ProposalStatistic> GetNumProposalsTopics() {
            return this.proposalTopicRepository.getNumProposalsTopics();
    }

    @GetMapping("/proposals/statistics/districts")
    public List<ProposalStatistic> GetNumProposalsDistricts() {
            return this.proposalLocationRepository.getNumProposalsDdistricts();
    }

    @GetMapping("/proposals/statistics/tags")
    public List<ProposalStatistic> GetNumProposalsTags() {
            return this.proposalTagRepository.getNumProposalsTags();
    }

    // Get a Single element
    @GetMapping("/proposals/{id}")
    public Proposal getItemById(@PathVariable(value = "id") String id) {
        return proposalRepository.findById(id)
                .orElseThrow(() -> new ResourceNotFoundException("Proposal", "id", id));
    }
}
```

Figura 27 – Controlador API Rest de Propuestas